Thinking of ReLU networks as a **hash table that looks up linear associations** is a surprisingly powerful mental model. It's not literally how they're implemented, but structurally it captures several real properties of piecewise-linear networks. Here are some deeper thoughts to sharpen that viewpoint and also clarify its limits.

---

1. What the "hash table" analogy gets exactly right

A ReLU network partitions input space into regions defined by activation patterns:

- Each neuron contributes a binary decision (active/inactive).
- Across a layer (or multiple layers), these combine into a **bit pattern**.
- Each distinct pattern corresponds to a **linear function**.

So in effect:

- Input → activation pattern (a "key")
- Key → linear map (the "value")

That *is* structurally equivalent to a hash table lookup followed by a linear transformation.

Two important refinements:

- The "hash" is **data-dependent and learned**, not random (after training).
- The lookup is **implicit**, not indexed in memory—it's realized by the network's computation graph.

---

2. It's not just a hash table — it's a *locality-sensitive* one

Your point about binarized random projections is crucial:

- At initialization, ReLU layers behave like **random hyperplane partitions**.
- The resulting bit patterns act like a **locality-sensitive hash (LSH)**.

This gives the system a very useful property:

> Nearby inputs tend to activate similar neuron subsets → select similar linear maps.

So unlike a classical hash table:

- It is **not arbitrary addressing**.

- It is **geometrically structured addressing**.

This makes it closer to:

- LSH tables
- Decision trees with soft boundaries
- Voronoi-like partitioning of space

---

3. The “value” is not stored independently

In a real hash table:

- Each key maps to an independently stored value.

In a ReLU network:

- The “linear maps” are **heavily shared compositions of weights**.

This has two major consequences:

(a) Massive compression

Instead of storing (2^n) independent linear maps:

- The network stores a **factorized representation** of exponentially many regions.

(b) Coupled updates

Training one region affects others because:

- They reuse the same weights.
- The system cannot freely memorize arbitrary mappings per region.

This is where your observation about **training resolving conflicts** becomes central.

---

4. Training = resolving collisions in a structured hash table

In a normal hash table:

- Collisions are undesirable.

In a ReLU network:

- Collisions are *inevitable and necessary*.

Different inputs may:

- Map to the same activation pattern, or
- Map to nearby patterns that share most weights

Training then becomes:

> Finding weights that make all “colliding” or “neighboring” regions agree on a consistent linear behavior.

This reframes generalization:

- The model generalizes well when **many regions can share a compatible linear explanation**.
- It overfits when it effectively creates too many nearly-independent regions.

---

5. Gating makes the hash table explicit

Your extension (multi-bit gating selecting weight blocks) pushes the analogy further:

- Instead of implicit selection via ReLU masks,
- You get **explicit indexing into a set of linear maps**.

This moves the model toward:

- Mixture-of-experts
- Conditional computation
- Sparse routing networks

At the extreme, you approach:

> A learned hash table of linear operators with structured parameter sharing.

---

6. Depth = composing hash lookups

In deep networks:

- Each layer re-hashes the representation.

- Each layer selects a new linear transform.

So the computation becomes:

A *composition of context-dependent linear maps*

This is more powerful than a single lookup because:

- Later “hashes” depend on earlier transformations.
- The system builds **hierarchical addressing**.

You can think of it as:

- First hash: coarse partition
- Later hashes: refine within that region

---

7. Where the analogy breaks down

The hash-table view is insightful, but incomplete:

(a) No discrete storage boundary

There is no explicit table:

- The mappings are **distributed across weights**.

(b) Regions are not independent

Unlike hash entries:

- Regions are geometrically adjacent and algebraically coupled.

(c) Boundaries are learned and movable

Hash functions are usually fixed:

- Here, the “hash function” (hyperplanes) is learned.

(d) It’s continuous under the hood

Even though activation patterns are discrete:

- Small parameter changes move boundaries continuously.

---

8. Why this view is still very useful

Despite the limitations, this perspective gives several powerful intuitions:

(1) Capacity comes from region count

- More gating bits → more regions → more linear maps

(2) Generalization comes from sharing

- Good models reuse structure across many regions

(3) Depth builds compositional addressing

- Not just more regions, but more *structured reuse*

(4) It explains "memorization vs structure"

- Pure memorization = independent regions
- Good learning = shared linear structure across regions

---

9. A useful mental summary

A concise way to think about it:

> A ReLU network is a **learned, locality-sensitive, highly compressed hash table** that maps inputs to **context-dependent linear transformations**,
> where training enforces **consistency across overlapping regions**.

---

10. One provocative implication

Your viewpoint suggests a design axis that's still underexplored:

- Instead of:
  - Dense layers + implicit gating (ReLU)
- Move toward:
  - **Explicit gating + modular linear blocks**

This leads directly to architectures like:

- Mixture-of-experts (but finer-grained)
- Neural decision forests with linear leaves
- Block-sparse conditional computation

And potentially:

> Systems where *the primary object is the routing (hashing), not the weights*.